Alcom-FT Technical Report Series
ALCOMFT-TR-01-96

External Memory BFS on Undirected Graphs with Bounded Degree

Ulrich Meyer*

Abstract

We give the first external memory algorithm for breadth-first search (BFS) which achieves $o(n)$ I/Os on arbitrary undirected graphs with n nodes and maximum node degree d. Let M and $B > d$ denote the main memory size and block size, respectively. Using $\mathrm{Sort}(x) = \Theta(\frac{x}{B} \cdot \log_{M/B} \frac{x}{B})$, our algorithm needs $\mathcal{O}(\frac{n}{\gamma \cdot \log_d B} + \mathrm{Sort}(n \cdot B^\gamma))$ I/Os and $\mathcal{O}(n \cdot B^\gamma)$ external space for an arbitrary parameter $0 < \gamma \leq 1/2$. The result carries over to BFS, depth-first search (DFS) and single source shortest paths (SSSP) on undirected planar graphs with arbitrary node degrees.

1 Introduction

We use the standard I/O model of [1], which counts accesses to a disk of potentially infinite size using the parameters M for the memory size and B for the block size where $B \leq M/2$. Let $\mathrm{Sort}(x) = \Theta(\frac{x}{B} \cdot \log_{M/B} \frac{x}{B})$ denote the number of I/Os needed to sort x items, and $\mathrm{Scan}(x) = \lceil \frac{x}{B} \rceil$ the number of I/Os required to transfer x items between contiguous disk positions and internal memory. Given a graph G with n nodes and m edges the model applies when $M < n \leq m$.

The best known external memory algorithms for breadth-first search (BFS), depth-first search (DFS) and single-source shortest paths (SSSP) on general undirected graphs still require $\Omega(n)$ I/Os, even if the graphs are planar and/or have bounded node degrees: $\mathcal{O}(n + \frac{m}{n} \cdot \mathrm{Sort}(n))$ I/Os for BFS [4], $\mathcal{O}(n + \frac{m}{B} \cdot \log \frac{n}{B})$ I/Os for SSSP [3], and $\mathcal{O}(\min\{\frac{n \cdot m}{M \cdot B} + n\ ,\ (n + \frac{m}{B}) \cdot \log \frac{n}{B}\})$ I/Os for DFS [6]. Better algorithms are known for special graph classes, see [6] for an overview. Furthermore, there is an $\mathcal{O}(\mathrm{Sort}(n))$ I/O algorithm for SSSP on undirected planar graphs G with bounded degree [2]. However, it requires a BFS-tree for G as part of the input.

New Results. We show how a modification of the BFS algorithm of Munagala and Ranade [4] can take advantage of a redundant graph representation. For arbitrary undirected graphs with maximum node degree $d < B$ we obtain an $\mathcal{O}(\frac{n}{\gamma \cdot \log_d B} + \mathrm{Sort}(n \cdot B^\gamma))$ I/O algorithm with external space requirement $\mathcal{O}(n \cdot B^\gamma)$ for an arbitrary parameter $0 < \gamma \leq 1/2$. The extended graph representation can be built within the above bounds. Using results of [2] we obtain the first $o(n)$ I/O bound for BFS, DFS and SSSP on undirected planar graphs with arbitrary node degrees.

2 Preliminaries

Let $L(t)$ denote the set of nodes in the BFS level t, and let $|L(t)|$ be the number of nodes in $L(t)$. The BFS algorithm of Munagala and Ranade [4] builds $L(t)$ as follows: let $A := N(L(t-1))$ be the set of neighbor vertices of nodes in $L(t-1)$. $N(L(t-1))$ is created by $|L(t-1)|$ accesses to the adjacency lists of all nodes in $L(t-1)$, thus incurring $\mathcal{O}(|L(t-1)| + \frac{|N(L(t-1))|}{B})$ I/Os. After removing duplicates from A the algorithm computes $L(t) := A \setminus \{L(t-1) \cup L(t-2)\}$. This can be done via sorting and scanning using $\mathcal{O}(\lceil \mathrm{Sort}(|N(L(t-1))| + |L(t-1)| + |L(t-2)|) \rceil)$ I/Os. Since $\sum_t |N(L(t))| = \mathcal{O}(m)$ and $\sum_t |L(t)| = \mathcal{O}(n)$, the whole algorithm of [4] incurs $\mathcal{O}(n + \mathrm{Sort}(n+m))$ I/Os.

3 The New Algorithm

In a preprocessing step we replace the adjacency lists kept for each node v by its k-neighborhood $N_k(v)$, i.e., the tree of nodes reachable via k edges from v without duplicate elimination[1]. Thus, the node representation is blown up by a factor of at most d^k where d denotes the maximum node degree in the graph. We choose k maximal with $k \cdot d^k \leq B^\gamma$, $0 < \gamma \leq 1/2$, hence $k = \Theta(1/\gamma \cdot \log_d B)$. Instead of loading adjacency lists for the construction of $N(L(t-1))$, the required edges are extracted from the enriched node information in $L(t-1)$ via sorting and scanning. However, this works for at most k levels, then new k-neighborhoods have to be accessed. Loading $N_k(v)$ for each node v in $L(t-1)$ in order to construct $L(t)$ might incur much unstructured I/O if $|L(t-1)|$ happens to be large. Therefore, we load the k-neighborhood for the nodes of the level $L(i)$, $t - k/2 \leq i < t$, such that $L(i)$ stores the smallest

*Max-Planck-Institut für Informatik, Stuhlsatzenausweg 85, 66123 Saarbrücken, Germany. www.mpi-sb.mpg.de/~umeyer/
Work partially supported by the IST Programme of the EU under contract number IST-1999-14186 (ALCOM-FT).

[1] Similar storing schemes already appear in [5]. However, they do not solve BFS but try to store a graph such that a single path which is given online by an adversary can be followed in an I/O-efficient way.

number of nodes. The new pieces of information suffice until the algorithm reaches level $L(i+k)$, i.e., at least level $L(t+k/2)$ can be reached without reload.

Thus, reloading occurs after every $k/2$ new levels the earliest. Due to the selection rule of least filled levels, reloading x k-neighborhoods saves arbitrary I/Os for at least $(k/2-1)\cdot x$ nodes (because each skipped level stores at least x nodes), i.e., $N_k(v)$ is accessed for at most $\mathcal{O}(n/k)$ nodes v over the whole course of the algorithm. Hence, we have eliminated $\Omega(n)$ arbitrary accesses to adjacency lists by $\mathcal{O}(n/k)$ accesses to data blocks of size $\Theta(d^k) < B$ each.

Note that if the number of nodes in a level $L(t)$ is at most k then the total amount of data kept in level $L(t+i)$ is at most $\mathcal{O}(k \cdot d^k \cdot d^i) = \mathcal{O}(B)$ for $i \leq k$. Therefore, from $L(t+2)$ till $L(t+k)$ all required data fits internally, no external sorting is needed, and the result for those levels can be written back to disk in one step after $L(t+k)$ is reached. Hence, the total number of non-blocked I/Os due to rounding while sorting and scanning is bounded by $\mathcal{O}(n/k)$. The remaining I/O costs for sorting and scanning in order to create the new levels add up to $\mathcal{O}(\mathrm{Sort}(d^k \cdot n))$ I/Os.

The preprocessing needs $\Theta(k)$ sorting phases, the amount of data increases with each phase by a factor of d. Altogether it takes $\mathcal{O}(\mathrm{Sort}(d^k \cdot n))$ I/Os. Hence, the complete algorithm requires $\mathcal{O}(\frac{n}{k} + \mathrm{Sort}(d^k \cdot n))$ I/Os. Using $k = \Theta(1/\gamma \cdot \log_d B)$ and $k \cdot d^k \leq B^\gamma$ yields the desired bound.

Theorem 1. *BFS on undirected graphs with maximum node degree $d < B$ needs $\mathcal{O}(\frac{n}{\gamma \cdot \log_d B} + \mathrm{Sort}(n \cdot B^\gamma))$ I/Os and $\mathcal{O}(n \cdot B^\gamma)$ external space for any $0 < \gamma \leq 1/2$.*

Now we turn to planar graphs. We assume that the edges are ordered according to an embedding in the plane. Then the SSSP algorithm of Arge, Brodal and Toma [2] takes $\mathcal{O}(\mathrm{Sort}(n))$ I/Os on undirected planar graphs G with maximum constant node degree if a BFS-tree for G is part of the input. Using our new algorithm for the construction of the BFS tree, Theorem 1 carries over to SSSP for undirected planar graphs with bounded degree.

For arbitrary node degrees we modify the initial graph G to G' as follows: each node v with degree $g > 3$ is replaced by a ring of g nodes $v_1, \ldots, v_g$ with zero weights edges. Each ring node v_i is connected to the former i-th neighbor of v (or its respective ring node) by an edge of the former weight. Thus, each node v_i in G' has the same weighted distance from the source as v in G. The transformation can be done with $\mathcal{O}(\mathrm{Sort}(n))$ I/Os. G' is still planar, embedded, and contains $n' = \mathcal{O}(n)$ nodes. Now we apply Theorem 1 to compute the BFS-tree of G' (ignore weights) and compute the SSSP of G' with algorithm of [2]. A solution for BFS is obtained by assigning unit cost edge weights in G and running the above algorithm.

Paper [2] also provides an algorithm which separates a planar graph G with bounded constant node degree into a set S of $\mathcal{O}(\mathrm{Sort}(n) + n/\sqrt{B})$ separator nodes and $\Theta(n/B)$ subgraphs of $\mathcal{O}(B)$ nodes (and edges) each. Again, the separator algorithm needs $\mathcal{O}(\mathrm{Sort}(n))$ I/Os provided that a BFS tree for G is given along with the input. Thus, the separation can be obtained within the bound of Theorem 1.

Given such a graph partition, a simple DFS algorithm for planar graphs with bounded degree keeps the visited-nodes information for each subgraph in $\mathcal{O}(1)$ blocks that are updated whenever it enters or leaves the respective subgraph. This can happen at most $\mathcal{O}(|S|)$ times since the degree of each separator node is constant and therefore, the total number of edges crossed between subgraphs and separator nodes is $\mathcal{O}(|S|)$. By the same argument there are at most $\mathcal{O}(|S|)$ arbitrary accesses for the separator nodes. Hence, the I/O bound is dominated by the construction of the BFS tree.

The case of arbitrary node degrees is handled again by the above graph transformation. The resulting DFS numbers $1, \ldots, n'$ from G' have to be thinned out to $1, \ldots, n$ by ignoring for each introduced ring all but the very first visit. This can be done by sorting and scanning.

Theorem 2. *BFS, DFS and SSSP on undirected planar graphs needs $\mathcal{O}(\frac{n}{\gamma \cdot \log_3 B} + \mathrm{Sort}(n \cdot B^\gamma))$ I/Os and $\mathcal{O}(n \cdot B^\gamma)$ external space for any $0 < \gamma \leq 1/2$.*

References

[1] A. Aggarwal and J.S. Vitter. The input/output complexity of sorting and related problems. *Communications of the ACM*, pages 1116–1127, 1988.

[2] L. Arge, G. Brodal, and L. Toma. On external-memory MST, SSSP and multi-way planar graph separation. In *Proc. 8th Scand. Workshop on Alg. Theory*, LNCS. Springer, 2000.

[3] V. Kumar and E. J. Schwabe. Improved algorithms and data structures for solving graph problems in external memory. In *Proc. 8th Symp. on Parallel and Distrib. Processing*, pages 169–177. IEEE, 1996.

[4] K. Munagala and A. Ranade. I/O-complexity of graph algorithms. In *Proc. 10th Symp. on Discrete Algorithms*, pages 687–694. ACM-SIAM, 1999.

[5] M. H. Nodine, M. T. Goodrich, and J. S. Vitter. Blocking for external graph searching. *Algorithmica*, 16(2):181–214, 1996.

[6] J. S. Vitter. External memory algorithms and data structures: Dealing with MASSIVE DATA. Online Version: `http://www.cs.duke.edu/~jsv/` , 2000.